Programowanie strukturalne - Wykład 9

Napisy (łańcuchy znaków)

Wyświetlanie napisów cd.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char buffer[20];
    int a=5;
    int b=7;
    sprintf(buffer,"%5d+%5d=%5d",a,b,a+b);
    printf("%s",buffer);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char buffer[20];
    int a=5;
    int b=7;
    snprintf(buffer,20*sizeof(char),"%5d+%5d=%5d",a,b,a+b);
    printf("%s",buffer);
    return 0;
}
```

Formaty

https://gist.github.com/pjastr/100599c74e34cd814f671daccd0668

https://gist.github.com/pjastr/37e259e243a7b9d26e9c45d528c84c

https://gist.github.com/pjastr/8c059084a44781c9407ee6ab0a3c5e

https://gist.github.com/pjastr/1285dbb398d7cfed9d372b7ed292d5

https://gist.github.com/pjastr/f59613feebb3900634cbe41658ae2be

Uwagi do pierwszego listingu (1)

1. %a i %A: Wypisanie liczby zmiennoprzecinkowej w zapisie szesnastkowym (szesnastkowe współczynniki, p-stwo p). %a używa małych liter (‘a’-‘f’), a %A dużych liter (‘A’-‘F’).
2. %c: Wypisanie pojedynczego znaku. Wartość przekazana jako argument jest interpretowana jako liczba całkowita, ale jest wyświetlana jako znak wg kodu ASCII.
3. %d: Wypisanie liczby całkowitej zapisanej jako dziesiętna.
4. %e i %E: Wypisanie liczby zmiennoprzecinkowej w notacji naukowej. %e używa małej litery ‘e’, a %E używa dużej litery ‘E’.

Uwagi do pierwszego listingu (2)

5. `%f`: Wypisanie liczby zmiennoprzecinkowej w postaci dziesiętnej.
6. `%g` i `%G`: Wypisanie liczby zmiennoprzecinkowej w skróconej notacji naukowej lub dziesiętnej, w zależności od wartości. `%g` używa małej litery ‘e’ dla notacji naukowej, a `%G` używa dużej litery ‘E’.
7. `%i`: Wypisanie liczby całkowitej zapisanej jako dziesiętna (to samo co `%d`).
8. `%o`: Wypisanie liczby całkowitej zapisanej jako ósemkowa.

Uwagi do pierwszego listingu (3)

9. %p: Wypisanie adresu wskaźnika.
10. %s: Wypisanie ciągu znaków.
11. %u: Wypisanie liczby całkowitej bez znaku zapisanej jako dziesiętna.
12. %x i %X: Wypisanie liczby całkowitej zapisanej jako szesnastkowa. %x używa małych liter (‘a’-‘f’), a %X dużych liter (‘A’-‘F’).

Uwagi do drugiego listingu (1)

1. %d: Wypisanie liczby całkowitej w postaci dziesiętnej.
2. % d: Wypisanie liczby całkowitej z dodatkową spacją przed liczbą dodatnią, aby zachować wyrównanie z liczbami ujemnymi.
3. %2d: Wypisanie liczby całkowitej z minimalną szerokością pola równą 2. Jeśli liczba ma mniej niż 2 cyfry, zostanie wypełniona spacjami.
4. %7d: Wypisanie liczby całkowitej z minimalną szerokością pola równą 7. Jeśli liczba ma mniej niż 7 cyfr, zostanie wypełniona spacjami.

Uwagi do drugiego listingu (2)

5. `%7.2d`: Wypisanie liczby całkowitej z minimalną szerokością pola równą 7 oraz minimalną liczbą cyfr wynoszącą 2. Jeśli liczba ma mniej niż 7 cyfr, zostanie wypełniona spacjami, a jeśli ma mniej niż 2 cyfry, zostanie wypełniona zerami.
6. `%07d`: Wypisanie liczby całkowitej z minimalną szerokością pola równą 7. Jeśli liczba ma mniej niż 7 cyfr, zostanie wypełniona zerami.
7. `%07.2d`: Wypisanie liczby całkowitej z minimalną szerokością pola równą 7 oraz minimalną liczbą cyfr wynoszącą 2. Jeśli liczba ma mniej niż 7 cyfr, zostanie wypełniona zerami.
8. `%-8d`: Wypisanie liczby całkowitej z minimalną szerokością pola równą 8, wyrównane do lewej. Jeśli liczba ma mniej niż 8 cyfr, zostanie wypełniona spacjami.

Uwagi do trzeciego listingu (1)

1. `%f`: Wypisanie liczby zmiennoprzecinkowej w postaci dziesiętnej.
2. `%lf`: Wypisanie liczby zmiennoprzecinkowej typu `double` w postaci dziesiętnej. W rzeczywistości, `%f` i `%lf` działają tak samo w przypadku funkcji `printf`, ponieważ argumenty typu `float` są promowane do `double` przy przekazywaniu do funkcji o zmiennej liczbie argumentów, takich jak `printf`.
3. `%5.2f`: Wypisanie liczby zmiennoprzecinkowej z minimalną szerokością pola równą 5 oraz 2 miejscami po przecinku.

Uwagi do trzeciego listingu (2)

4. `%3.1f`: Wypisanie liczby zmiennoprzecinkowej z minimalną szerokością pola równą 3 oraz 1 miejscem po przecinku.
5. `%10.3f`: Wypisanie liczby zmiennoprzecinkowej z minimalną szerokością pola równą 10 oraz 3 miejscami po przecinku.
6. `%+10.3f`: Wypisanie liczby zmiennoprzecinkowej z minimalną szerokością pola równą 10, 3 miejscami po przecinku oraz zawsze z wyświetlanym znakiem liczby (+ lub -).

Uwagi do trzeciego listingu (3)

7. `%010.2f`: Wypisanie liczby zmiennoprzecinkowej z minimalną szerokością pola równą 10, 2 miejscami po przecinku, wypełniona zerami.
8. `%010.1e` i `%010.1E`: Wypisanie liczby zmiennoprzecinkowej w notacji naukowej z minimalną szerokością pola równą 10, 1 miejscem po przecinku, wypełniona zerami. `%010.1e` używa małej litery ‘e’, a `%010.1E` używa dużej litery ‘E’.

Uwagi do czwartego listingu (1)

1. `%x`: Wypisanie liczby całkowitej w postaci szesnastkowej, używając małych liter (‘a’-‘f’).
2. `%X`: Wypisanie liczby całkowitej w postaci szesnastkowej, używając dużych liter (‘A’-‘F’).
3. `%#x`: Wypisanie liczby całkowitej w postaci szesnastkowej z prefiksem `0x`, używając małych liter (‘a’-‘f’).

Uwagi do czwartego listingu (2)

4. `%.0f`: Wypisanie liczby zmiennoprzecinkowej z zaokrągleniem do liczby całkowitej (0 miejsc po przecinku).
5. `%#.0f`: Wypisanie liczby zmiennoprzecinkowej z zaokrągleniem do liczby całkowitej oraz z wymuszonym wyświetleniem separatora dziesiętnego (kropki) mimo braku miejsc po przecinku.

Uwagi do piątego listingu (1)

1. `%s`: Wypisanie łańcucha znaków.
2. `%20s`: Wypisanie łańcucha znaków z minimalną szerokością pola równą 20. Jeśli łańcuch ma mniej niż 20 znaków, zostanie wypełniony spacjami z lewej strony.
3. `%-20s`: Wypisanie łańcucha znaków z minimalną szerokością pola równą 20, wyrównane do lewej. Jeśli łańcuch ma mniej niż 20 znaków, zostanie wypełniony spacjami z prawej strony.

Uwagi do piątego listingu (2)

4. `%20.5s`: Wypisanie łańcucha znaków z minimalną szerokością pola równą 20 oraz maksymalną długością łańcucha równą 5. Jeśli łańcuch ma mniej niż 20 znaków, zostanie wypełniony spacjami z lewej strony; jeśli ma więcej niż 5 znaków, zostanie obcięty do 5 znaków.
5. `%-20.5s`: Wypisanie łańcucha znaków z minimalną szerokością pola równą 20, wyrównane do lewej oraz maksymalną długością łańcucha równą 5. Jeśli łańcuch ma mniej niż 20 znaków, zostanie wypełniony spacjami z prawej strony; jeśli ma więcej niż 5 znaków, zostanie obcięty do 5 znaków.

Typ `wchar_t`

https://en.wikibooks.org/wiki/C_Programming/wchar.h

https://en.cppreference.com/w/c/language/string_literal

```c
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>

int main()
{
    wchar_t buffer[20];
    fgetws(buffer,20,stdin);
    fputws(buffer,stdout);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>

int main()
{
    wchar_t buffer[20];
    wscanf(L"%s",buffer);
    wprintf(L"%s",buffer);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>

int main()
{
    setlocale(LC_ALL,"");
    wchar_t buffer[20];
    fgetws(buffer,20,stdin);
    fputws(buffer,stdout);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>

int main()
{
    setlocale(LC_ALL, "pl_PL.UTF-8");
    wchar_t buffer[20];
    fgetws(buffer,20,stdin);
    fputws(buffer,stdout);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>

int main()
{
    wchar_t buffer[20];
    int a=3;
    int b=4;
    swprintf(buffer,20*sizeof(wchar_t),L"%d+%d=%d",a,b,a+b);
    wprintf(L"%s",buffer); // na linuxie %ls
    return 0;
}
```

Napisy a funkcje

```
1  int dlugosc(char*napis)
2  {
3      int temp=0;
4      while(*(napis++))
5      {
6          temp++;
7      }
8      return temp;
9  }
```

```
1  int dlugosc2(char napis[])
2  {
3      int temp=0;
4      for(int i=0;napis[i]!='\0';i++)
5      {
6          temp++;
7      }
8      return temp;
9  }
```

czy to możliwe?

```
1 void foo(const char*napis)
2 {
3     *napis='a';
4 }
```

```c
#include <stdio.h>
#include <stdlib.h>

char* foo()
{
    return "abc";
}

int main()
{
    printf("%s\n",foo());
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

char* foo()
{
    char * temp=(char*)malloc(sizeof(char)*10);
    temp[0]='w';
    temp[1]='$';
    temp[2]='a';
    temp[3]='\0';
    return temp;
}

int main()
{
    printf("%s\n",foo());
    return 0;
}
```

Tak nie robimy (!)

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  char* foo()
 5  {
 6      char temp[10];
 7      temp[0]='w';
 8      temp[1]='$';
 9      temp[2]='a';
10      temp[3]='\0';
11      return temp;
12  }
13
14  int main()
15  {
16      printf("%s\n",foo());
17      return 0;
18  }
```

Podsumowanie

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char tekst1[10]="abcde";
    printf("%Iu\n",sizeof(tekst1));
    printf("%p\n",tekst1);
    printf("%p\n",&tekst1);
    //tekst1="eee";
    //tekst1++;
    tekst1[2]='R';
    printf("%s\n",tekst1);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *tekst2="abcde";
    printf("%Iu\n",sizeof(tekst2));
    printf("%p\n",tekst2);
    printf("%p\n",&tekst2);
    tekst2="WERT";
    printf("%s\n",tekst2);
    tekst2++;
    //tekst2[2]='R';
    printf("%s\n",tekst2);
    return 0;
}
```

Funckje znakowe i łańuchowe

https://en.cppreference.com/w/c/string/byte

https://en.cppreference.com/w/c/string/wide

Funkcje znakowe

1. `isalnum` - sprawdza, czy znak jest alfanumeryczny (cyfra lub litera).
2. `isalpha` - sprawdza, czy znak jest literą (alfabetycznym).
3. `islower` - sprawdza, czy znak jest małą literą.
4. `isupper` - sprawdza, czy znak jest dużą literą.
5. `isdigit` - sprawdza, czy znak jest cyfrą.

6. `isxdigit` - sprawdza, czy znak jest cyfrą szesnastkową.
7. `iscntrl` - sprawdza, czy znak jest znakiem sterującym.
8. `isgraph` - sprawdza, czy znak jest znakiem graficznym (znak drukowany, ale nie jest spacją).
9. `isspace` - sprawdza, czy znak jest znakiem białym (białe znaki to: spacja, tabulacja, nowa linia, powrót karetki, itp.).
10. `isblank` (C99) - sprawdza, czy znak jest pustym znakiem (spacja lub tabulacja).

11. `isprint` - sprawdza, czy znak jest znakiem drukowanym (włączając spację).
12. `ispunct` - sprawdza, czy znak jest znakiem interpunkcyjnym.
13. `tolower` - konwertuje znak na małą literę, jeśli jest to możliwe.
14. `toupper` - konwertuje znak na dużą literę, jeśli jest to możliwe.

Bibliografia

- Stephen Prata, Jezyk C. Szkoła programowania. Wydanie VI, Wyd. Helion, 2016.
- https://cybersecurity.umcs.lublin.pl/wp-content/uploads/kmazur/PP2017/, dostęp online 10.04.2020.
- Richard Reese, Wskaźniki w języku C, Wydawnictwo Helion 2014.